Natural Language Processing- Text classification with movie reviews

File Edit View Insert Runtime Tools Help All changes saved

Comment Share M

+ Code + Text

RAM Disk Editing

```
[28] from __future__ import absolute_import, division, print_function, unicode_literals

     import tensorflow as tf
     from tensorflow import keras

     import numpy as np

     import matplotlib.pyplot as plt
```

```
[29] print('Tensorflow: {}\nNumpy: {}'.format(tf.__version__, np.version.version))
```

```
Tensorflow: 2.9.2
Numpy: 1.21.6
```

Load the IMDB dataset

```
[30] imdb = keras.datasets.imdb
     (train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

About the data

```
[31] print('Training data: {}, labels: {}'.format(len(train_data), len(train_labels)))
```

```
Training data: 25000, labels: 25000
```

```
[32] print(train_data[0])
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 20
```

```
[33] len(train_data[0]), len(train_data[1])
```

```
(218, 189)
```

Converting integers back to words

```
[34] word_index = imdb.get_word_index()
```

```
[35] word_index = {k:(v+3) for k,v in word_index.items()}
     word_index['<PAD>'] = 0
     word_index['<START>'] = 1
     word_index['<UNK>'] = 2
     word_index['<UNUSED>'] = 3
```

```
[36] reverse_word_index = dict([(value, key) for  (key, value) in word_index.items()])
```

```
[37] def decode_review(text):
         return ' '.join([reverse_word_index.get(i, '?') for i in text])
```

```
[38] decode_review(train_data[0])
```

```
'<START> this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being there robert <UNK> is an amazing actor and now the same being director <UNK> father came fro
m the same scottish island as myself so i loved the fact there was a real connection with this film the witty remarks throughout the film were great it was just brilliant so much that i bought the film as soon as it was released for <UNK> and would r
ecommend it to everyone to watch and the fly fishing was amazing really cried at the end it was so sad and you know what they say if you cry at a film it must have been good and this definitely was also <UNK> to the two little boy's that played the <
UNK> of norman and paul they were just brilliant children are often left out of the <UNK> list i think because the stars that play them all grown up are such a big profile for the whole film but these children are amazing and should be praised for wh
a...'
```

Prepare the data

Since the movie reviews must be the same length, we will use the pad_sequences function to standardize the lengths.

```
[39] train_data = keras.preprocessing.sequence.pad_sequences(train_data, value=word_index['<PAD>'], padding='post', maxlen=256)
     test_data = keras.preprocessing.sequence.pad_sequences(test_data, value=word_index['<PAD>'], padding='post', maxlen=256)
```

```
[40] len(train_data[0]), len(train_data[1])
```

```
(256, 256)
```

```
[41] print(train_data[0])
```

Prepare the data

Since the movie reviews must be the same length, we will use the pad_sequences function to standardize the lengths.

```
[39] train_data = keras.preprocessing.sequence.pad_sequences(train_data, value=word_index['<PAD>'], padding='post', maxlen=256)
     test_data = keras.preprocessing.sequence.pad_sequences(test_data, value=word_index['<PAD>'], padding='post', maxlen=256)
```

```
[40] len(train_data[0]), len(train_data[1])
```

```
(256, 256)
```

```
print(train_data[0])
```

```
[  1  14  22  16  43 530 973 1622 1385  65 458 4468  66 3941
   4 173  36 256   5  25 100  43 838 112  50 670   2   9
  35 480 284   5 150   4 172 112 167   2 336 385  39   4
 172 4536 1111  17 546  38  13 447   4 192  50  16   6 147
2025  19  14  22   4 1920 4613 469   4  22  71  87  12  16
  43 530  38  76  15  13 1247   4  22  17 515  17  12  16
 626  18   2   5  62 386  12   8 316   8 106   5   4 2223
5244  16 480  66 3785  33   4 130  12  16  38 619   5  25
 124  51  36 135  48  25 1415  33   6  22  12 215  28  77
  52   5  14 407  16  82   2   8   4 107 117 5952  15 256
   4   2   7 3766   5 723  36  71  43 530 476  26 400 317
  46   7   4   2 1029  13 104  88   4 381  15 297  98  32
2071  56  26 141   6 194 7486  18   4 226  22  21 134 476
  26 480   5 144  30 5535  18  51  36  28 224  92  25 104
   4 226  65  16  38 1334  88  12  16 283   5  16 4472 113
 103  32  15  16 5345  19 178  32   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0]
```

Build the model

```
[42] VOCAB_SIZE = 10000
```

```
[43] model = keras.Sequential()
```

```
[44] model.add(keras.layers.Embedding(VOCAB_SIZE, 16))
     model.add(keras.layers.GlobalAveragePooling1D())
     model.add(keras.layers.Dense(16, activation=tf.nn.relu))
     model.add(keras.layers.Dense(1, activation=tf.nn.sigmoid))
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 16)          160000

global_average_pooling1d_1   (None, 16)              0
(GlobalAveragePooling1D)

dense_2 (Dense)              (None, 16)              272

dense_3 (Dense)              (None, 1)               17

=================================================================
Total params: 160,289
Trainable params: 160,289
Non-trainable params: 0
_________________________________________________________________
```

Compile the model

```
[46] model.compile(optimizer='adam',
                   loss='binary_crossentropy',
                   metrics=['accuracy'])
```

Create validation set

```
[47] x_val = train_data[:10000]
     partial_x_train = train_data[10000:]
```

```
[48] y_val = train_labels[:10000]
     partial_y_train = train_labels[10000:]
```

Training the mode

```
history = model.fit(partial_x_train, partial_y_train, epochs=40, batch_size=512, validation_data=(x_val, y_val), verbose=1)
```

```
Epoch 12/40
30/30 [==============================] - 0s 11ms/step - loss: 0.3773 - accuracy: 0.8765 - val_loss: 0.3975 - val_accuracy: 0.8542
Epoch 13/40
30/30 [==============================] - 0s 10ms/step - loss: 0.3560 - accuracy: 0.8851 - val_loss: 0.3775 - val_accuracy: 0.8592
Epoch 14/40
30/30 [==============================] - 0s 14ms/step - loss: 0.3286 - accuracy: 0.8905 - val_loss: 0.3611 - val_accuracy: 0.8639
Epoch 15/40
30/30 [==============================] - 0s 12ms/step - loss: 0.3087 - accuracy: 0.8960 - val_loss: 0.3479 - val_accuracy: 0.8676
Epoch 16/40
30/30 [==============================] - 0s 12ms/step - loss: 0.2922 - accuracy: 0.8998 - val_loss: 0.3375 - val_accuracy: 0.8687
Epoch 17/40
30/30 [==============================] - 0s 12ms/step - loss: 0.2765 - accuracy: 0.9048 - val_loss: 0.3272 - val_accuracy: 0.8749
Epoch 18/40
30/30 [==============================] - 0s 13ms/step - loss: 0.2631 - accuracy: 0.9093 - val_loss: 0.3195 - val_accuracy: 0.8764
Epoch 19/40
30/30 [==============================] - 0s 10ms/step - loss: 0.2508 - accuracy: 0.9130 - val_loss: 0.3133 - val_accuracy: 0.8782
Epoch 20/40
30/30 [==============================] - 0s 11ms/step - loss: 0.2398 - accuracy: 0.9170 - val_loss: 0.3075 - val_accuracy: 0.8800
Epoch 21/40
30/30 [==============================] - 0s 10ms/step - loss: 0.2292 - accuracy: 0.9207 - val_loss: 0.3029 - val_accuracy: 0.8811
Epoch 22/40
30/30 [==============================] - 1s 17ms/step - loss: 0.2197 - accuracy: 0.9241 - val_loss: 0.2993 - val_accuracy: 0.8822
Epoch 23/40
30/30 [==============================] - 0s 16ms/step - loss: 0.2106 - accuracy: 0.9271 - val_loss: 0.2955 - val_accuracy: 0.8822
Epoch 24/40
30/30 [==============================] - 0s 12ms/step - loss: 0.2026 - accuracy: 0.9310 - val_loss: 0.2927 - val_accuracy: 0.8833
Epoch 25/40
30/30 [==============================] - 0s 11ms/step - loss: 0.1946 - accuracy: 0.9335 - val_loss: 0.2917 - val_accuracy: 0.8836
Epoch 26/40
30/30 [==============================] - 0s 9ms/step - loss: 0.1872 - accuracy: 0.9369 - val_loss: 0.2887 - val_accuracy: 0.8841
Epoch 27/40
30/30 [==============================] - 0s 10ms/step - loss: 0.1801 - accuracy: 0.9410 - val_loss: 0.2883 - val_accuracy: 0.8841
Epoch 28/40
30/30 [==============================] - 0s 11ms/step - loss: 0.1738 - accuracy: 0.9437 - val_loss: 0.2870 - val_accuracy: 0.8843
Epoch 29/40
30/30 [==============================] - 0s 12ms/step - loss: 0.1675 - accuracy: 0.9459 - val_loss: 0.2861 - val_accuracy: 0.8844
Epoch 30/40
30/30 [==============================] - 0s 11ms/step - loss: 0.1614 - accuracy: 0.9493 - val_loss: 0.2856 - val_accuracy: 0.8857
Epoch 31/40
30/30 [==============================] - 0s 13ms/step - loss: 0.1557 - accuracy: 0.9507 - val_loss: 0.2856 - val_accuracy: 0.8862
Epoch 32/40
30/30 [==============================] - 0s 11ms/step - loss: 0.1500 - accuracy: 0.9533 - val_loss: 0.2864 - val_accuracy: 0.8859
Epoch 33/40
30/30 [==============================] - 0s 13ms/step - loss: 0.1452 - accuracy: 0.9559 - val_loss: 0.2862 - val_accuracy: 0.8857
Epoch 34/40
30/30 [==============================] - 0s 11ms/step - loss: 0.1399 - accuracy: 0.9580 - val_loss: 0.2871 - val_accuracy: 0.8865
Epoch 35/40
30/30 [==============================] - 0s 14ms/step - loss: 0.1351 - accuracy: 0.9603 - val_loss: 0.2880 - val_accuracy: 0.8860
Epoch 36/40
30/30 [==============================] - 0s 13ms/step - loss: 0.1305 - accuracy: 0.9619 - val_loss: 0.2896 - val_accuracy: 0.8862
Epoch 37/40
30/30 [==============================] - 0s 11ms/step - loss: 0.1261 - accuracy: 0.9643 - val_loss: 0.2911 - val_accuracy: 0.8850
Epoch 38/40
30/30 [==============================] - 0s 14ms/step - loss: 0.1219 - accuracy: 0.9659 - val_loss: 0.2929 - val_accuracy: 0.8860
Epoch 39/40
30/30 [==============================] - 0s 11ms/step - loss: 0.1183 - accuracy: 0.9659 - val_loss: 0.2941 - val_accuracy: 0.8852
```

Evaluation

```
[50]  Show code
```

```
782/782 [==============================] - 3s 3ms/step - loss: 0.3146 - accuracy: 0.8740
[0.3146378397941589, 0.8740400075912476]
```

```
[51] history_dict = history.history
     history_dict.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
[52] accuracy = history_dict['accuracy']
     val_accuracy = history_dict['val_accuracy']
     loss = history_dict['loss']
     val_loss = history_dict['val_loss']
```

```
[53] epochs = range(1, len(accuracy) + 1)
```

```
plt.xlabel('Epochs')

plt.plot(epochs, accuracy, 'g--', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'g', label='Validation accuracy')

plt.plot(epochs, loss, 'r--', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')

plt.legend()
plt.show()
```